Thomas Vu - 571563

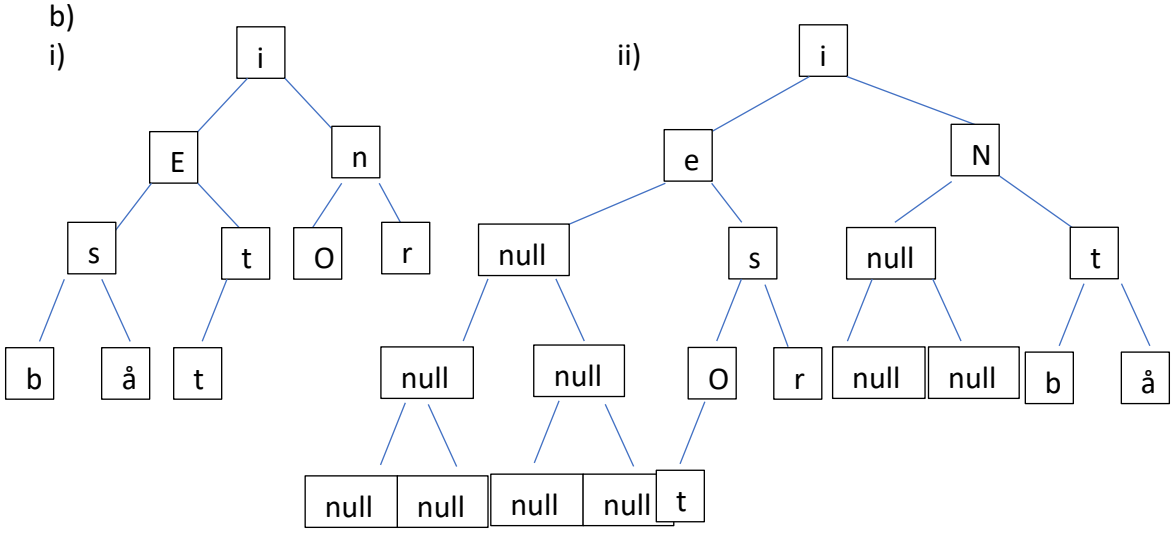
Dat102 Oblig 5

Oppgave 1:

a) Et binært tre er et tre hvor hver node maks 2 barn hver
Høyden til et binært tre er lengden på den lengste veien i treet. Fra rot til blad.
Fullt binært tre: Et tre som har maksimalt antall noder. Alle nivåene i treet er helt fylt opp med noder.
Komplett binært tre: Et tre som på alle nivåer, unntatt muligens det nederste, er helt fulle med noder. På nederste nivå skal alle nodene ligge så langt til venstre som mulig.

b)

i)



ii)

c) Pre-orden:
Tre 1: i-E-s-b-å-t-t-n-O-r
Tre 2: i-e-s-O-t-r-N-t-b-å
In-orden:
Tre 1: b-s-å-E-t-t-i-O-n-r
Tre 2: e-t-O-s-r-i-N-b-t-å
Post-orden:
Tre 1: b-å-s-t-t-E-O-r-n-i
Tre 2: t-O-r-s-e-b-å-t-N-i
Nivå-orden:
Tre 1: i-E-n-s-t-O-r-b-å-t
Tre 2: i-e-N-s-t-O-r-b-å-t

Oppgave 2:

c) $c*\log_2(1024) = 21.27$
c=2.127

```
Gjennomsnittshøyden til trærne: 21.27
```

$2.127*\log_2(8192) = 27.651$

```
Gjennomsnittshøyden til trærne: 29.43
```

Thomas Vu - 571563

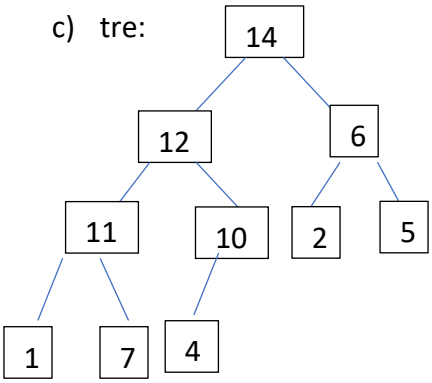
Oppgave 3:

a) En heap er en type binær tre som har tilleggs egenskaper. Vi har minimumsheap og maksimumsheap. Minimumsheap er et binært og komplett minimumstre og maksimumsheap er et binært og komplett maksimumstre.

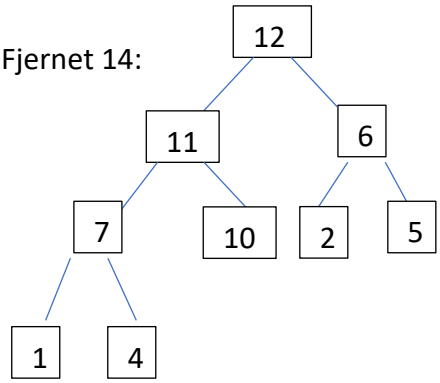
   Et binærtre kalles et minimumstre hvis verdien i hver indre node er mindre enn eller lik verdiene i nodens barn, og det blir kalt maksimumstre hvis verdien i hver indre node er større enn eller lik verdiene i nodens barn.

b) Man kan se at A ikke er maks haug med en gang siden 9 er mindre enn 15, og for at det skal være en makshaug så må nodene være større enn eller lik nodens barn.
   B og C er makshaug siden alle nodene er større enn nodens barn, og den største verdien er roten.

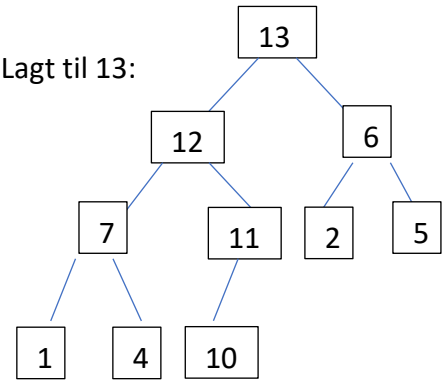
c) tre:



Fjernet 14:

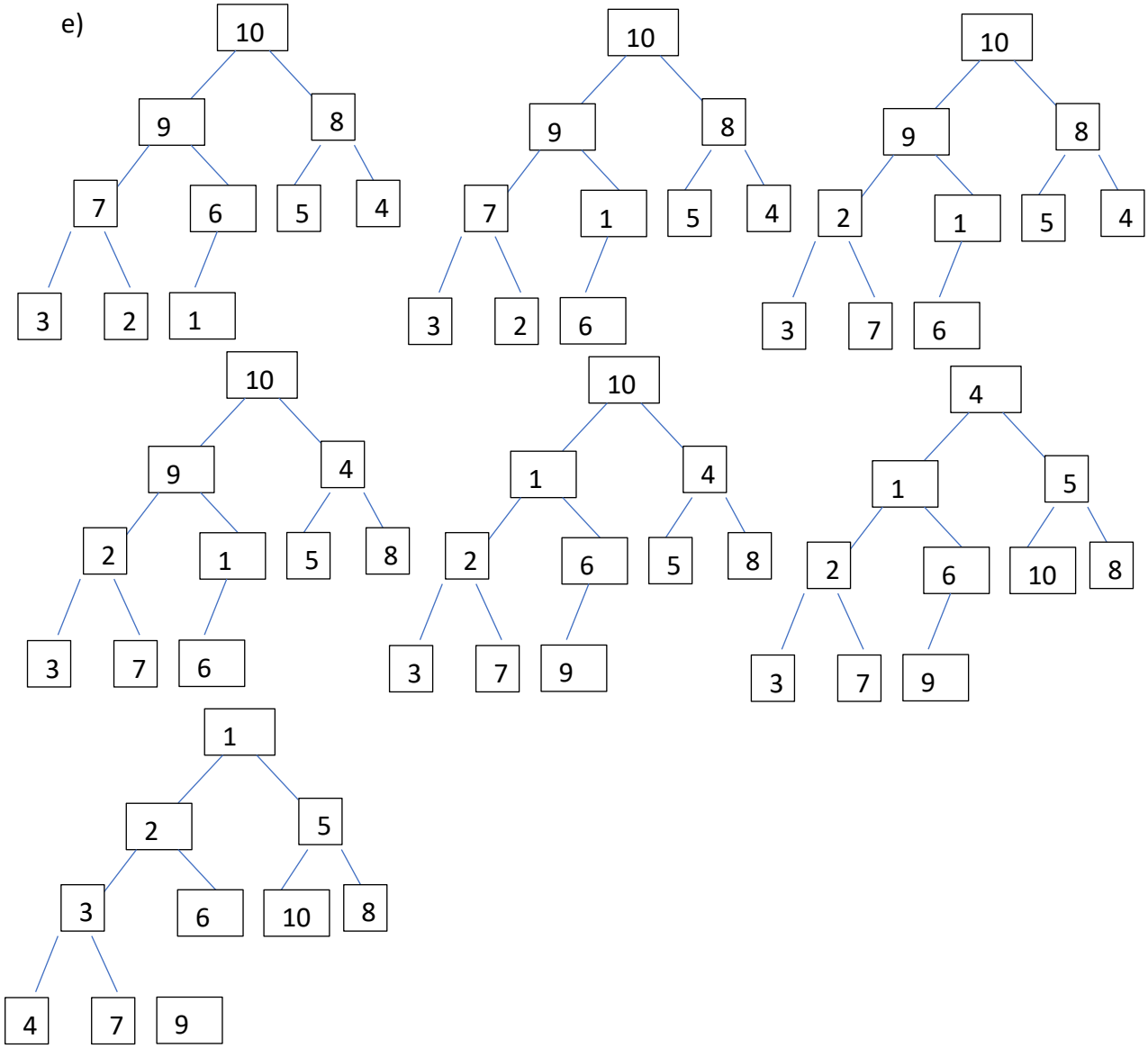


Lagt til 13:



d) Man setter inn elementet på den siste plassen i tabellen og sjekker om det elementet er større enn elementets forelder. Hvis det nye elementet som blir satt inn er større, så vil disse to bytte plass. Når den har bytta plass så sjekker den igjen helt forelder noden er større eller til noden treffer roten. På denne måten så blir elementene sortert.

e)



f)

```
<terminated> KlientHaug (1) [Java Application] /Library/Java/JavaVirt
Verdiene i tabellen er n�:
1 10 2 18 54 33 30 300 200 100

Haugen i sortert rekkefoelge:
1  2  10  18  30  33  54  100  200  300
```

Thomas Vu - 571563

Oppgave 4:

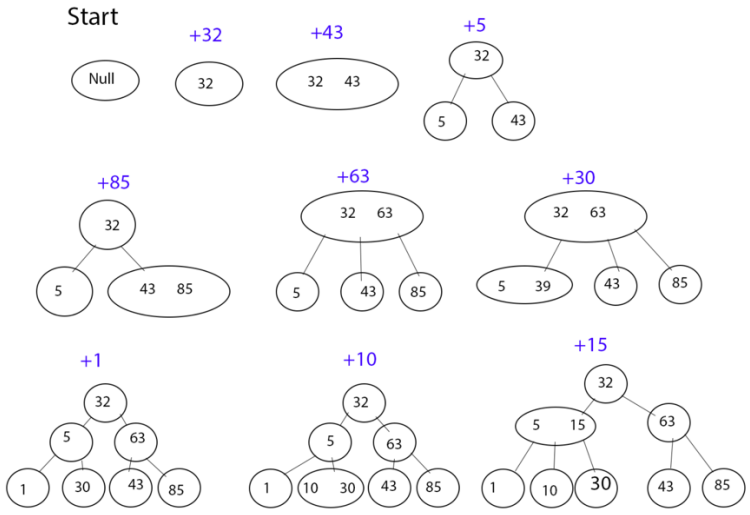
a) Et 2-3 tre er et flervekssøketre der hver node har null, to eller tre barn. Ingen noder har et barn, dette sikrer oss at vi ikke får lange lineære grener i treet. Alle bladene er på samme nivå, dermed er treet balansert.
2-3 trær er mer effektive til å søke enn BSTrær fordi det værste tilfelle av høyde for et 2-3 tre er O(log(n)).

b) En 2-node inneholder ett element og har enten ingen eller to barn. Elementene i venstre undertre er mindre enn elementet i 2-noden. Elementene i høyre undertre er større eller lik elementet i 2-noden.

En 3-node inneholder to elementer, ett kalt minste og ett kalt største. En 3-node har enten ingen barn eller tre barn.
Hvis en 3-node har barn så er elementene i venstre undertre mindre enn minste.
Elementene i det midterste undertre er større eller lik minste og mindre enn største.
Elementene i høyre undertreet er større eller lik største elementet.

c) Sammenligner 42 med roten 45. 42<45. Går ned til venstre.
Sammenligner 42 med 30. 42>30. Går videre ned til høyre
Sammenligner 42 med 35 først. 42>35. Går videre til høyre i noden.
Sammenligner 42 med 40. 42>40. Siden dette er en bladnode, så er det ikke noen steder å gå. Konkluderer med at 42 ikke finnes.

d)



Thomas Vu - 571563

Oppgave 5:

a)

Kø	Besøkt
c	c
cef	e
efbd	f
fb	b
bd	d
da	a
d	
tom	

b)

Kø	Besøkt
c	c
ef	f
ebe	e
ebbd	d
ebbab	b
ebba	a
ebb	
eb	
e	
tim	

c)

V = 6 og E = 8

	a	b	c	d	e	f
a				T		
b				T	T	T
c					T	T
d	T	T			T	
e		T	T	T		T
f		T	T		T	

a→d

b→d->e->f

c→e->f

d→a->b->e

e→d->b->c->f

f→e->b->c

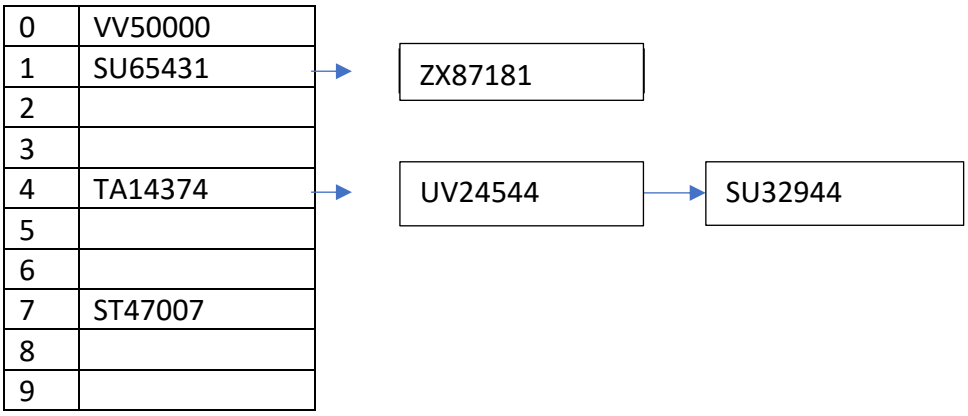
Thomas Vu - 571563

Oppgave 6:

a) Når man bruker hashing så blir elementene vi vil lagre, lagret i en hashtabell. Lokasjonene er basert på en hashfunksjon som for eksempel første bokstaven i et navn. Hver lokasjon i hashtabellen er kalt en celle eller en bøtte.
Når to nøkler gir samme resultat i hashfunksjonen så oppstår en kollisjon. Opphoping av data er resultater av kollisjoner. Når flere nøkkelverdier blir avbildet, vil dette fenomenet forsterkes.
En god hashfunksjon vil fordele elementene uniformt over hele hashtabellen. En perfekt hashfunksjon gir ingen kollisjoner.

b)

0	VV50000
1	SU65431
2	ZX87181
3	
4	TA14374
5	UV24544
6	SU32944
7	ST47007
8	
9	



c)

ab:

s(0)=97

s(1)=98

$s(0)*31^{(2-1)}+s(1)*31^{(2-2)}=3105$

s(0)=49

s(1)=50

s(2)=51

$s(0)*31^{(3-1)}+s(1)*31^{(3-2)}+s(2)*31^{(3-3)}=48690$

d)

De gir ikke samme utskrift på grunn av equals() metoden på de to studentobjektene ikke gir true som resultat, så de har forskjellige hashcoder. Man kan få samme utskrift om man gjør studentnummeret om til String og deretter bruler hashCode().